

Программирование Python

Структуры данных. Классы

Кафедра ИВТ и ПМ
ЗабГУ

2018

План

Прошлые темы

Пользовательские структуры данных

- Концепция записи

- Запись как кортеж

- Классы

- Работа с наборами объектов

- Классы внутри классов

Outline

Прошлые темы

Пользовательские структуры данных

- Концепция записи

- Запись как кортеж

- Классы

- Работа с наборами объектов

- Классы внутри классов

Прошлые темы

- ▶ Что такое тип?
- ▶ Какие простые типы есть в Python?
- ▶ Какие составные типы есть в Python?

Outline

Прошлые темы

Пользовательские структуры данных

Концепция записи

Запись как кортеж

Классы

Работа с наборами объектов

Классы внутри классов

Логически связанные данные

Иногда требуется работать одновременно с несколькими отдельными но логически связанными переменными.

Например координаты точки на плоскости представляются двумя переменными.

Это вызывает большие неудобства если таких логически связанных переменных становится много. Например описания погоды (температура, облачность, осадки) в определённом городе, в определённый день.

Неудобства проявляются особенно сильно если нужно создавать ещё одну группу таких переменных для хранения данных, например для другого города.

Outline

Прошлые темы

Пользовательские структуры данных

Концепция записи

Запись как кортеж

Классы

Работа с наборами объектов

Классы внутри классов

Запись

Логически такие наборы данных можно описывать в виде набора полей.

Предположим требуется хранить данные о погодных условиях (температура, облачность, осадки) в определённом городе, в полдень на определённую дату.

Данные о погоде:

Город

ДеньГода

Температура

Облачность

Осадки

Такое представление логически сгруппированных данных будем называть *записью*.



ВЫПИСЬ ИЗ МЕТРИЧЕСКОЙ КНИГИ, ЧАСТЬ ПЕРВАЯ, О РОДИВШИХСЯ, ЗА ГОДАХ.

Вік розрахунок	Підпис і дата	Забір, ім'я, статево і фамілія і кожного з дітей/дочок	Забір, ім'я, статево і фамілія кожної/кожних	Кто совершил таинство крещения	Родительство дети запис по ж дним
41. 25. 31	Марія	Сестра Ново-Горішівки прикладної Святицької Княгині Мико- лаїв Синайської і законної сес- тери Елизавети Михайлова, оба православныя.	Сестра Михайловская Святи- цька Павла Анисимови Михайлівська сестра Ново-Кем- ского Святицького сестра Екатерина Степановна Ке- рловская.	Святицький сестра Ми- хайловская Павел Михайлівський от діа- кона Павла Муратовиць	
1914. 25. 31	1914. 25. 31	Сіз метрическая книга по всел согласна с копією метрической книги за- писки в канцелярии духовной консистории, г. 14, канцелярия переписки сестры Ново-Горішівки, что удостоверяет подлинность и легальность Святицькой Княгини Павла. Получено в канцелярии			



Концепция записей

4.07.12	Ковалевская И.С.	9000	Всего тисл. 9000	Коллекция
07.12	Завалько А.Г.	2500	Фасель сох. 8	Завалько
07.12	Беленко А.А.	7000	Портрет Тютчева, 1881 г.	Беленко
07.12	Козлов И.Б.	1919	одна тисля, 1881 г.	Козлов
07.12	Козлов И.Б.	283,5	звезда Косенцев, три в 50.	Козлов
1.08	Козлов И.Б.	4000	звезда Косенцев	Козлов
1.08.12	Мухомов И.	4000	звезда Косенцев	Мухомов
4.06.12	Завалько А.Г.	16250	Мухомов, 1881 г.	Завалько
4.06.12	Савельев В.П.	1645	Одна тисля, 1881 г.	Савельев
4.06.12	Тютчев А.П.	4399	звезда Косенцев, 1881 г.	Тютчев
1.06.12	Тютчев А.П.	805	звезда Косенцев, 1881 г.	Тютчев
4.06.12	Тютчев А.П.	-11-	-11-11-11-11-	-11-
4.06.12	Савельев В.П.	130	звезда Косенцев, 1881 г.	Савельев
4.06.12	Тютчев А.П.	1500	звезда Косенцев, 1881 г.	Тютчев
4.06.12	Тютчев А.П.	378	звезда Косенцев, 1881 г.	Тютчев
25.07.12	Ленинградская С.А.	3188	звезда Косенцев, 1881 г.	Ленинградская
5.07.12	Ленинградская С.А.	2000	звезда Косенцев, 1881 г.	Ленинградская
5.07.12	Ленинградская С.А.	1195	звезда Косенцев, 1881 г.	Ленинградская

Запись

Запись (record) — тип данных, набор значений различных типов.

Запись состоит из **полей**. Поле как было отмечено может быть представлено отдельным типом. В поле в том числе может быть другой записью.

Например поле *Дата* представляет собой отдельную запись состоящую из трёх полей.

Данные о погоде:

Город

Дата

год

месяц

день

Температура

Облачность

Осадки

Запись

В языках программирования записи могут быть представлены разными способами.

В Python запись можно представить с помощью одного из составных типов данных: *кортежа, списка или словаря*.

Другой способ представления записи - тип данных **класс (class)**.

Хотя понятия класса включает в себя ещё и методы (операции производимые с данными), его можно использовать только для хранения данных.

Outline

Прошлые темы

Пользовательские структуры данных

Концепция записи

Запись как кортеж

Классы

Работа с наборами объектов

Классы внутри классов

Записи с помощью кортежей

Представим дату (год, месяц, день) как кортеж.

```
date1 = (2018, 03, 14)
```

#Теперь можно обращаться к отдельным элементам кортежа

```
date1[0]  # 2018
```

```
date1[2]  # 14
```

Создавать списки из кортежей

```
dates = []
```

```
for i in range(10):
```

```
    d = randint(2000, 2018), \
```

```
        randint(1, 12), \
```

```
        randint(1, 28)
```

```
    dates += [ d ]
```

Outline

Прошлые темы

Пользовательские структуры данных

Концепция записи

Запись как кортеж

Классы

Работа с наборами объектов

Классы внутри классов

Классы и объекты

Класс — составной тип данных, который может быть описан программистом.

Объект - экземпляр класса; *переменная* типа класс.

Классы и объекты

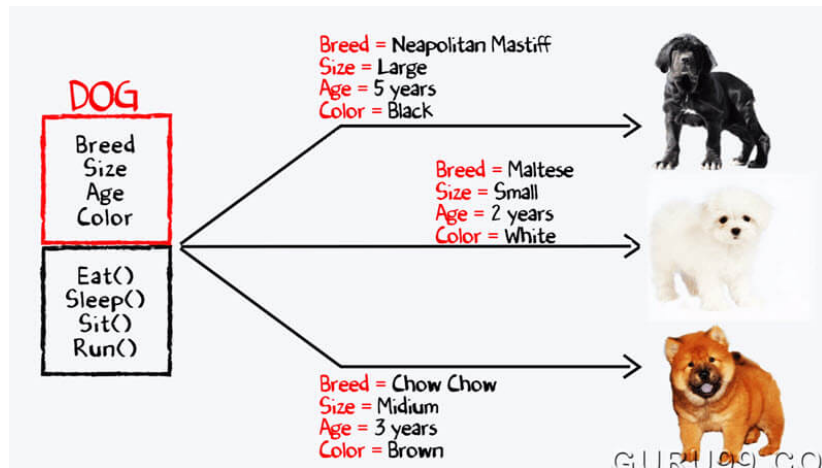
Классы могут включать в себя переменные других типов - **поля** (свойства).

Кроме того, класс может включать в себя функции - **методы**.

Класс как тип данных представляет собой только набор полей которым не заданы конкретные значений и набор методов.

Во время создания объекта (объявления переменной типа класс) этим полям задаются конкретные значения.

Классы и объекты



Класс как контейнер данных

Представим запись *Дата* в виде класса (создадим новый тип данных).

```
class Date:
    day = 1
    month = 1
    year = 1
```

В Python нельзя объявить поле, только определить: привести идентификатор и значение.

Когда будет создана переменная типа *Date* её поля уже будут содержать указанные при определении класса значения.

Если нужно отличать переменную с заполненными значениями полей, от аналогичной незаполненными то в качестве начальных значений используют `None`.

Класс как контейнер данных

Создание переменной описанного типа Date.

Переменная типа класс называется **объектом** или **экземпляром** класса.

```
my_birthday = Date()  
othder_date = Date()  
d1 = Date()
```

```
type( d1 )  # __main__.Date
```

Каждый описанный класс представляет собой отдельный тип данных.

С технической точки зрения имя класса в python включает ещё и имя пространства имён. В приведённом примере пространство имён называется `__main__`¹

¹Имена переменных начинающиеся и заканчивающиеся символа подчёркивания играют роль служебных ("для внутреннего пользования"), и непосредственное использование таких имён не рекомендуется

Класс как контейнер данных

Чтобы получить доступ к полям используется **селектор** - оператор "."(точка).

```
my_birthday = Date()
```

```
my_birthday.day = 31
```

```
my_birthday.month = 1
```

```
my_birthday.year = 1956
```

Класс как контейнер данных

В Python любой тип являются классом, в том числе простые типы.

Однако часто с точки зрения программиста работа с переменными встроенных типов (int, float, tuple и т.д.) выглядит точно так же как и с обычными переменными (в других языках).

Класс как контейнер данных

```
my_birthday = Date()
```

```
my_birthday.day = 31
```

```
my_birthday.month = 1
```

```
my_birthday.year = 1956
```

```
# отдельные экземпляры класса независимы
```

```
d = Date()
```

```
d.year = 1984
```

```
print( my_birthday.year )    # 1956
```

```
print( d.year )    # 1984
```

Таким образом для логически связанных наборов данных следует создавать либо классы либо организовывать их с помощью встроенных типов: списков, кортежей, словарей.

Такой подход позволяет логически организовать данные, уменьшить количество отдельных переменных.

Точка на плоскости:

```
# точка 1  
x1, y1 = 0,0  
  
# точка 2  
x2, y2 = 0,0  
  
# 2 точки - 4 переменных
```

```
class Point2D:
```

```
    x = 0
```

```
    y = 0
```

```
# точка 1
```

```
p1 = Point2D()
```

```
# точка 2
```

```
p1 = Point3D()
```

```
# 2 точки - 2 переменные
```

Конструктор

Если используется кортеж вместо класса, то возможно задать значения элементов в одну строчку:

```
x,y = 7.2, -0.5
```

Однако если использовать этот способ для класса Point2D то будет создан кортеж

```
p = Point3D()
```

```
p = 7.2, -0.5
```

```
type( p )  # -> tuple
```

Конструктор

Чтобы решить эту проблему в класс, помимо *полей* нужно добавить *методы*, т.е. функции класса.

Метод который используется для создания объекта называется **конструктором**.

В Python такой метод должен всегда называться `__init__`

```
class Point2D:
    x = 0
    y = 0

    def __init__(self, x,y):
        self.x = x
        self.y = y
```

```
# теперь можно создать объект так
p = Point2D( 7.2, -0.5 )
```

Конструктор

- ▶ Конструктор является специальным методом и вызывается не так как остальные.
- ▶ Конструктор вызывается во время создания объекта (переменной)
- ▶ Конструктор инициализирует объект - задаёт начальные значения полей.
- ▶ Конструктор без параметров задаёт полям те значения, которые были приведены при их описании.

Конструктор

- ▶ Конструктор при описании внутри класса должен быть назван `__init__`, однако при его вызове должно быть указано имя класса:

```
def Point2D():  
    x,y, = 0, 0  
    def __init__(self, x,y):  
        self.x = x  
        self.y = y
```

```
p1 = Point2D(10, 2)
```

- ▶ Конструктор вызывается во время создания объекта
`p1 = Point2D(10, 2)`

Конструктор

```
class Point2D:  
    x = 0  
    y = 0  
  
    def __init__(self, x,y,z):  
        self.x = x  
        self.y = y
```

Поля `x` и `y` класса `Point2D` не входят в область видимости метода `__init__`, однако им должны задаваться значения.

В Python к полям класса изнутри методов класса можно обращаться через переменную `self`.

Эта переменная представляет *сам объект*, описывается как первый параметр каждого метода, но при вызове метода передаётся неявно.

Другие методы

```
class Point2D:
    x = 0
    y = 0

    def r(self):
        """ Возвращает расстояние от точки до начала координат """
        return ( self.x**2 + self.y**2 ) ** 0.5

p = Point2D()
p.x = 3
p.y = 4

p.r()    # 5
```

Неявную передачу `self` можно представлять как передачу параметра p в функцию $r()$, только этот параметр записывается не в скобках, а перед именем функции с точкой на конце.

Другие методы

Использование self похоже на явную передачу параметра в отдельную функцию.

```
class Point2D:
    x = 0
    y = 0
    # метод
    def r(self):
        """ Возвращает расстояние от точки до начала координат """
        return ( self.x**2 + self.y**2 ) ** 0.5
    # функция
    def R( point ):
        return ( point.x**2 + point.y**2 ) **2

p = Point2D()
p.x = 3
p.y = 4

p.r()    # 5

R( p )   # 5
```

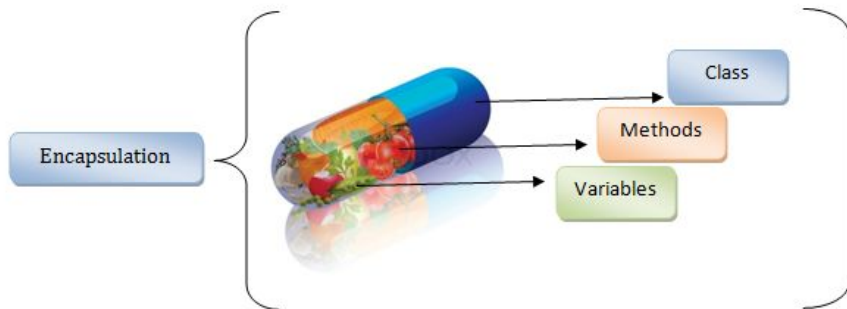

Инкапсуляция

Таким образом, помимо данных классы могут содержать ещё и методы работы с этими данными.

Такое объединение данных и методов к классу называется **инкапсуляцией**.

При этом подразумевается, что методы могут контролировать целостность и непротиворечивость данных.

Инкапсуляция



Пример

Класс "Точка" с двумя полями: x и y ; двумя методами: конструктором `__init__`

```
class Point2D:
    x = 0
    y = 0

    #
    def __init__(self, x,y,z):
        self.x = x
        self.y = y

    def r(self):
        """ Возвращает расстояние от точки до начала координат """
        return ( self.x**2 + self.y**2 ) ** 0.5
```

```
p1 = Point2D( 10.5, -1.02 )
p2 = Point2D( 0, 42 )
```

```
p1.r()  # 10.55
```

```
p2.r()  # 42
```

- ▶ **Класс** - это тип данных определяемый программистом.
- ▶ Класс может содержать как данные (**поля**) так и описывать операции над этими данными (**методы**).
- ▶ **Объект** - переменная имеющая тип определённого класса.

```
p = Point2D()
```

```
# p - объект  
# Point2D - класс
```

- ▶ Объекты одного и того же класса имеют одинаковые методы и одинаковые поля, но данные содержащиеся в полях могут отличаться.
- ▶ Для обращения объекта к самому себе (своим полям и методам) используется зарезервированное имя **self**
- ▶ Все переменные в Python, включая простые типы, являются объектами.

Пользовательские (созданные программистом) объекты нельзя непосредственно выводить на экран, записывать введенные с клавиатуры данные непосредственно в объект.

Непосредственная запись в файл и чтение не рекомендуется, потому, что объект может содержать кроме данных ещё и методы.

Однако, для объекта можно написать методы, которые помогут выполнить вышеперечисленные операции.

Пример

```
class Point2D:
    x = 0
    y = 0
    # ...

    # преобразует массив байт в x,y
    def from_bytes(self, b):
        self.x, self.y = unpack('ff', b)

    # преобразует данные объекта в массив байт
    def to_bytes(self):
        return pack('ff', self.x, self.y)

p1 = Point2D(10.5, -1.02)
b = p1.to_bytes()
p1.from_bytes(b)
f = open('file.in', 'w+b')
f.write( b )
f.seek(0)
p1.from_bytes(f.read(8))
f.close()
```

Outline

Прошлые темы

Пользовательские структуры данных

Концепция записи

Запись как кортеж

Классы

Работа с наборами объектов

Классы внутри классов

Пример

Создание списка из объектов

```
class Point2D:
    x = 0
    y = 0

    def __init__(self, x, y):
        self.x = x
        self.y = y
```

```
points = [ Point2D(1.23, -7), Point2D(42, -7.123),
            Point2D(42, 43),   Point2D(0, 123) ]
```


Пример

Создание списка из точек со случайными координатами

```
points = []  
  
for i in range(10):  
    p = Point2D (random()*100-50, random()*100-50)  
    points += [ p ]
```

Вывод объектов на экран

Вывод объектов на экран.

Объект произвольного класса может быть непригоден для непосредственного вывода на экран.

Но если поля класса могут быть легко напечатаны, то можно вывести их значения.

```
for p in points:  
    print( "{:.2f} {:.2f}".format(p.x, p.y), end=", ")
```

или

```
for i in range(len(points)):  
    print( "{:.2f} {:.2f}".format(points[i].x,  
                                  points[i].y), end=", ")
```

Вывод объектов на экран

Вывод объектов на экран.

В предыдущем примере для каждого объекта из списка пришлось выполнять одну и ту же операцию: преобразование его полей в строковый тип для представления на экране.

Такую операцию логично поместить внутрь класса.

```
class Point2D:
    x = 0
    y = 0
    #...

    def to_str(self):
        """ Возвращает строковое представление точки """
        return "({:.2f}, {:.2f})".format(self.x, self.y)
```

Вывод объектов на экран

Выводить объекты на экран, если у них есть метод, который представляет их содержимое в текстовом виде.

```
for p in points:  
    print( p.to_str(), end=", ")
```

или

```
for i in range(len(points)):  
    print( points[i].to_str(), end=", ")
```

Методы или функции?

Стоит стремиться к реализации действий над классом внутри класса - в *методах*, а не в отдельных функциях, не принадлежащих классу.

Таким образом будет выполнен принцип **инкапсуляции** - объединение данных и методов работы с ними в классе.

Операции производимые же со множеством объектов, например списком, имеет смысл выносить в отдельные *функции*.

Пример

Поиск объекта в списке с определённым значениям поля

```
points = [ Point2D(1.23, -7), Point2D(42, -7.123),  
           Point2D(42, 43), Point2D(0, 123) ]  
  
y1 = 43  
l = None # номер искомой точки в списке  
for i in range(len(points)):  
    if points[i].y == y1:  
        l = i  
        break  
  
if l is not None:  
    print("Индекс искомого объекта в списке: ", l)  
else:  
    print("Объект в списке не найден")
```

Методы по работе

Тип данных = возможные значения + операции.

Так как класс - тип данных который создаёт программист, то и операции по работе с этим классом нужно создавать самостоятельно.

Для таких классов методы сравнения ($<$, $>$, и т.п.), добавления, вычитания и проч не реализованы.

А результат операция сравнения $==$ будет Истина только при сравнении объекта с самим с собой, но не с объектом с такими же значениями полей.

Поиск класса в списке

Например операция сравнения (`==`) для класса `Point2D` не реализована программистом.

Следующий сниппет будет работать некорректно:

```
points = [ Point2D(1.23, -7), Point2D(42, -7.123),  
          Point2D(42, 43), Point2D(0, 123) ]
```

```
p1 = Point2D(42, 43)
```

```
l = None # номер искомой точки в списке  
for i in range(len(points)):  
    if points[i] == p1:  
        l = i  
        break
```

```
if l is not None:  
    print("Индекс искомого объекта в списке: ", l)  
else:  
    print("Объект в списке не найден")
```

```
# Объект в списке не найден
```


Поиск класса в списке

Корректный способ сравнения объектов типа Point2D.

```
points = [ Point2D(1.23, -7), Point2D(42, -7.123),  
           Point2D(42, 43), Point2D(0, 123) ]
```

```
p1 = Point2D(42, 43)
```

```
l = None # номер искомой точки в списке
```

```
for i in range(len(points)):
```

```
    if points[i].x == p.x and points[i].y == p.y:
```

```
        l = i
```

```
        break
```

```
if l is not None:
```

```
    print("Индекс искомого объекта в списке: ", l)
```

```
else:
```

```
    print("Объект в списке не найден")
```

```
# Индекс искомого объекта в списке: 2
```

Сортировка объектов

Если для данного класса имеет смысл операция сравнения, то экземпляры этого класса можно упорядочивать.

Сравнение "больше-меньше" имеет смысл для точек: например сравнение точек по отдельным координатам, или по их совокупности (расстояние от центра координат до данной точки)

Такую операцию сравнения также нужно реализовывать программисту.

Пример.

Сортировка объектов (точек) по первой координате от большей к меньшей (по убыванию)

```
points = [ Point2D(1.23, -7), Point2D(42, -7.123),  
           Point2D(42, 43),   Point2D(0, 123) ]  
  
# используем пузырьковую сортировку  
for i in range(len(points)-1):  
    for j in range(len(points)-i-1):  
        if points[j].x < points[j+1].x:  
            # обмен значений через кортежное присваивание  
            points[j], points[j+1] = points[j+1], points[j]  
  
for p in points:  
    print( p.to_str(), end=", ")  
# (42.00, -7.12), (42.00, 43.00), (1.23, -7.00), (0.00, 123.00),
```

Пример.

Сортировка объектов (точек) по расстоянию от центра координат.

```
class Point2D:
    # ...
    def r(self):
        return ( self.x**2 + self.y**2 )**0.5

points = [ Point2D(1.23, -7), Point2D(42, -7.123),
           Point2D(42, 43),   Point2D(0, 123) ]

# используем пузырьковую сортировку
for i in range(len(points)-1):
    for j in range(len(points)-i-1):
        if points[j].r() < points[j+1].r():
            # обмен значений через кортежное присваивание
            points[j], points[j+1] = points[j+1], points[j]

for p in points:
    print( p.to_str(), end=" ", )
# (0.00, 123.00), (42.00, 43.00), (42.00, -7.12), (1.23, -7.00)
```

Outline

Прошлые темы

Пользовательские структуры данных

Концепция записи

Запись как кортеж

Классы

Работа с наборами объектов

Классы внутри классов

Классы внутри классов

Для представления сложных объектов реального мира или абстракций используют классы которые могут включать в себя поля имеющие тип других классов.

Пример

Класс представляющий отрезок на плоскости.

```
class LineSegment:
    p1 = Point2D(0,0)    # точка 1
    p2 = Point2D(0,0)    # точка 2

    def __init__(self, x1,y1, x2,y2):
        self.p1.x = x1
        self.p1.y = y1
        self.p2.x = x2
        self.p2.y = y2
```

```
ls1 = LineSegment( 0.2,3, 5,5)
```

```
print( ls1.p1.x ) # 0.2
```

```
ls1.p2 = Point2D(33, 44)
```

```
print( ls1.p2.r() ) # 55
```

Ссылки и литература

Ссылка на слайды

github.com/VetrovSV/Programming